

Semantik von SPARQL

Thanh Tran Sebastian Rudolph

Institut AIFB · Karlsruhe Institute of Technology

Semantic Web Technologies 1 (WS12/13)

<http://semantic-web-grundlagen.de>

Foliensatz von M. Krötzsch. Die nichtkommerzielle Vervielfältigung, Verbreitung und Bearbeitung dieser Folien ist zulässig (→ Lizenzbestimmungen CC-BY-NC).

- 1 Einleitung und Motivation
- 2 Umwandlung von Anfragen in SPARQL-Algebra
- 3 Rechnen mit der SPARQL-Algebra
- 4 Zusammenfassung

- 1 Einleitung und Ausblick
- 2 XML und URIs
- 3 Einführung in RDF
- 4 RDF Schema
- 5 Logik – Grundlagen
- 6 Semantik von RDF(S)
- 7 OWL – Syntax und Intuition
- 8 OWL – Semantik und Reasoning
- 9 OWL 2 – Syntax und Semantik
- 10 SPARQL – Syntax und Intuition
- 11 Semantik von SPARQL

Letzte Vorlesung: SPARQL als Anfragesprache für RDF

```
PREFIX ex: <http://example.org/>
SELECT ?buch, ?autor WHERE
{ ?buch ex:VerlegtBei <http://springer.com/Verlag> .
  ?buch ex:Preis      ?preis .
  ?buch ex:Autor      ?autor
  FILTER (?preis < 35)
} ORDER BY ?preis LIMIT 10
```

Merkmale von SPARQL:

- Einfache, optionale und alternative Graphmuster
- Filter
- Ausgabeformate (SELECT, CONSTRUCT, ...)
- Modifikatoren (ORDER BY, LIMIT, ...)

Fragestellung für diese Vorlesung:

Wie genau ist die Semantik von SPARQL definiert?

Bisher lediglich informelle Darstellung von SPARQL

- Anwender: „Welche Antworten kann ich auf meine Anfrage erwarten?“
- Entwickler: „Wie genau soll sich meine SPARQL-Implementierung verhalten?“
- Hersteller: „Ist mein Produkt bereits SPARQL-konform?“

⇒ Formale Semantik schafft (hoffentlich) Klarheit . . .

Semantik formaler Logik (siehe Vorlesung 5):

- Modelltheoretische Semantik: Welche Interpretationen erfüllen eine Wissensbasis?
- Beweistheoretische Semantik: Welche Ableitungen aus einer Wissensbasis sind zulässig?
- ...

Semantik von Anfragesprachen (1)

Semantik formaler Logik (siehe Vorlesung 5):

- Modelltheoretische Semantik: Welche Interpretationen erfüllen eine Wissensbasis?
- Beweistheoretische Semantik: Welche Ableitungen aus einer Wissensbasis sind zulässig?
- ...

Semantik von Programmiersprachen:

- Axiomatische Semantik: Welche logischen Aussagen gelten für ein Programm?
- Operationale Semantik: Wie wirkt sich die Abbarbeitung eines Programms aus?
- Denotationelle Semantik: Wie kann ein Programm als Eingabe/Ausgabe-Funktion abstrakt dargestellt werden?

Was tun mit Anfragesprachen?

Semantik von Anfragesprachen (2)

Semantik von Anfragesprachen

Semantik von Anfragesprachen:

Anfragefolgerung (*query entailment*)

- Anfrage als Beschreibung zulässiger Anfrageergebnisse
- Datenbasis als Menge logischer Annahmen (Theorie)
- Ergebnis als logische Schlussfolgerung

Bsp.: OWL DL und RDF(S) als Anfragesprachen, konjunktive Anfragen

Semantik von Anfragesprachen (2)

Semantik von Anfragesprachen:

Anfragefolgerung (*query entailment*)

- Anfrage als Beschreibung zulässiger Anfrageergebnisse
- Datenbasis als Menge logischer Annahmen (Theorie)
- Ergebnis als logische Schlussfolgerung

Bsp.: OWL DL und RDF(S) als Anfragesprachen, konjunktive Anfragen

Anfragealgebra

- Anfrage als Rechenvorschrift zur Ermittlung von Ergebnissen
- Datenbasis als Eingabe
- Ergebnis als Ausgabe

Bsp.: Relationale Algebra für SQL, SPARQL-Algebra

Outline

- 1 Einleitung und Motivation
- 2 **Umwandlung von Anfragen in SPARQL-Algebra**
- 3 Rechnen mit der SPARQL-Algebra
- 4 Zusammenfassung

Übersetzung in SPARQL-Algebra

```
{ ?buch    ex:Preis    ?preis .  
  FILTER (?preis < 15)  
  OPTIONAL  
    { ?buch    ex:Titel    ?titel . }  
  { ?buch    ex:Autor    ex:Shakespeare . } UNION  
  { ?buch    ex:Autor    ex:Marlowe . }  
}
```

Semantik einer SPARQL-Anfrage:

- 1 Umwandlung der Anfrage in einen algebraischen Ausdruck
- 2 Berechnung des Ergebnisses dieses Ausdrucks

Übersetzung in SPARQL-Algebra: *BGP*

```
{ ?buch    ex:Preis    ?preis .  
  FILTER (?preis < 15)  
  OPTIONAL  
    { ?buch    ex:Titel    ?titel . }  
  { ?buch    ex:Autor    ex:Shakespeare . } UNION  
  { ?buch    ex:Autor    ex:Marlowe . }  
}
```

Erster Schritt: **Ersetzung einfacher Graph-Muster**

- Operator *BGP*
- gleichzeitig Auflösung von abgekürzten URIs

Übersetzung in SPARQL-Algebra: *BGP*

```
{ BGP (?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    { BGP (?buch <http://eg.org/Titel> ?titel.) }  
    { BGP (?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.) }  
  UNION  
    { BGP (?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.) }  
}
```

Erster Schritt: Ersetzung einfacher Graph-Muster

- Operator *BGP*
- gleichzeitig Auflösung von abgekürzten URIs

Übersetzung in SPARQL-Algebra: *Union*

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.)}  
  UNION  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.)}  
}
```

Zweiter Schritt: Zusammenfassung alternativer Graph-Muster

- Operator *Union*
- Bezug auf an UNION angrenzende Muster (\rightsquigarrow bindet stärker als Konjunktion)
- Klammerung mehrerer Alternativen wie in Vorlesung 9 besprochen (linksassoziativ)

Übersetzung in SPARQL-Algebra: *Union*

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.)}  
  
  UNION  
    {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.)}  
}
```

Zweiter Schritt: Zusammenfassung alternativer Graph-Muster

- Operator *Union*
- Bezug auf an UNION angrenzende Muster (\rightsquigarrow bindet stärker als Konjunktion)
- Klammerung mehrerer Alternativen wie in Vorlesung 9 besprochen (linksassoziativ)

Übersetzung in SPARQL-Algebra: *Union*

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
  Union ({BGP(?buch <http://eg.org/Autor>  
           <http://eg.org/Shakespeare>.)},  
         {BGP(?buch <http://eg.org/Autor>  
           <http://eg.org/Marlowe>.)})  
}
```

Zweiter Schritt: *Zusammenfassung alternativer Graph-Muster*

- Operator *Union*
- Bezug auf an `UNION` angrenzende Muster (\rightsquigarrow bindet stärker als Konjunktion)
- Klammerung mehrerer Alternativen wie in Vorlesung 9 besprochen (linksassoziativ)

Übersetzung in SPARQL-Algebra

$Join(M_1, M_2)$	konjunktive Verknüpfung von M_1 und M_2
$LeftJoin(M_1, M_2, F)$	optionale Verknüpfung von M_1 mit M_2 unter der Filterbedingung F
$Filter(F, M)$	Anwendung des Filterausdrucks F auf M
Z	Konstante für <i>leeren Ausdruck</i>

Übersetzung in SPARQL-Algebra

$Join(M_1, M_2)$	konjunktive Verknüpfung von M_1 und M_2
$LeftJoin(M_1, M_2, F)$	optionale Verknüpfung von M_1 mit M_2 unter der Filterbedingung F
$Filter(F, M)$	Anwendung des Filterausdrucks F auf M
Z	Konstante für <i>leeren Ausdruck</i>

Verbleibende Übersetzung schrittweise von innen nach außen:

- 1 Wähle ein innerstes gruppierendes Graph-Muster M
- 2 Entferne Filterausdrücke aus M ;
 $GF :=$ Konjunktion der Filterbedingungen
- 3 Initialisiere $G := Z$, und arbeite alle Teilausdrücke UA ab:
 - Falls $UA = \text{OPTIONAL } Filter(F, A)$: $G := LeftJoin(G, A, F)$
 - Ansonsten, falls $UA = \text{OPTIONAL } A$: $G := LeftJoin(G, A, \text{true})$
 - Sonst: $G := Join(G, UA)$
- 4 Falls GF nicht leer ist: $G := Filter(GF, G)$

Übersetzung in SPARQL-Algebra: *(Left)Join, Filter* (1)

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    {BGP(?buch <http://eg.org/Titel> ?titel.)}  
  Union({BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Shakespeare>.)},  
        {BGP(?buch <http://eg.org/Autor>  
          <http://eg.org/Marlowe>.)})  
}
```

Übersetzung in SPARQL-Algebra: *(Left)Join, Filter* (1)

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    Join(Z, BGP(?buch <http://eg.org/Titel> ?titel.))  
  Union(Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Shakespeare>)),  
    Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Marlowe>)))  
}
```

Übersetzung in SPARQL-Algebra: *(Left)Join, Filter* (2)

```
{ BGP(?buch <http://eg.org/Preis> ?preis.)  
  FILTER (?preis < 15)  
  OPTIONAL  
    Join(Z, BGP(?buch <http://eg.org/Titel> ?titel.))  
  Union(Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Shakespeare>)),  
    Join(Z, BGP(?buch <http://eg.org/Autor>  
    <http://eg.org/Marlowe>)))  
}
```

Übersetzung in SPARQL-Algebra: *(Left)Join, Filter* (2)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      Join(Z, BGP(?buch <http://eg.org/Preis> ?preis.)),  
      Join(Z, BGP(?buch <http://eg.org/Titel> ?titel.)),  
      true  
    ), Union(Join(Z, BGP(?buch <http://eg.org/Autor>  
      <http://eg.org/Shakespeare>)),  
      Join(Z, BGP(?buch <http://eg.org/Autor>  
      <http://eg.org/Marlowe>)))  
  )  
)
```

Operationen zur Darstellung der Modifikatoren:

<i>OrderBy</i> (G , Sortierangaben)	Sortiere Lösungen in Ergebnisliste
<i>Distinct</i> (G)	Entferne doppelte Lösungen aus Ergebnisliste
<i>Slice</i> (G , o , l)	Beschneide Ergebnisliste auf Abschnitt der Länge l ab Position o
<i>Project</i> (G , Variablenliste)	Beschränke alle Lösungen auf die angegebenen Variablen

Die Modifikator-Operationen werden in bestimmter Reihenfolge angewandt:

- 1 $G := \text{OrderBy}(G, \text{Sortieranweisungen})$, wenn `ORDER BY` mit diesen Sortieranweisungen verwendet wurde.
- 2 $G := \text{Project}(G, \text{Variablenliste})$, wenn das Format `SELECT` mit dieser Liste ausgewählter Variablen verwendet wurde.
- 3 $G := \text{Distinct}(G)$, wenn `DISTINCT` verwendet wurde.
- 4 $G := \text{Slice}(G, o, l)$, wenn Angaben „`OFFSET o`“ und „`LIMIT l`“ gemacht wurden. Standardwerte bei fehlender Angabe sind $o = 0$ und $l = \text{Länge von } G - o$.

- 1 Einleitung und Motivation
- 2 Umwandlung von Anfragen in SPARQL-Algebra
- 3 Rechnen mit der SPARQL-Algebra**
- 4 Zusammenfassung

Wie sind die Operationen der SPARQL-Algebra definiert?

Ausgabe:

- „Ergebnistabelle“ (Formatierung hier nicht relevant)

Eingabe:

- Angefragte RDF-Datenbasis
- Teilergebnisse von Unterausdrücken
- verschiedene Parameter je nach Operation

⇒ Wie sollen „Ergebnisse“ formal dargestellt werden?

SPARQL-Ergebnisse

Intuition: Ergebnisse kodieren Tabellen mit Variablenbelegungen

Ergebnis:

Liste von *Lösungen* (Lösungssequenz)

↪ jede Lösung entspricht einer Tabellenzeile

SPARQL-Ergebnisse

Intuition: Ergebnisse kodieren Tabellen mit Variablenbelegungen

Ergebnis:

Liste von *Lösungen* (Lösungssequenz)

↪ jede Lösung entspricht einer Tabellenzeile

Lösung:

Partielle Abbildung (Funktion)

- Definitionsbereich (Domäne): ausgewählte Menge von Variablen
- Wertebereich: URIs \cup leere Knoten \cup RDF-Literale

↪ Ungebundene Variablen sind solche, die von einer Lösung keinen Wert zugewiesen bekommen (*partielle* Funktion).

Wofür steht der „leere Ausdruck“ Z ?

Wofür steht der „leere Ausdruck“ Z ?

- Domäne: \emptyset (keine ausgewählten Ergebnisse)
- Lösungen: genau eine (es gibt eine Funktion mit leerem Wertebereich, aber nur eine)

\rightsquigarrow „Tabellen mit einer Zeile aber keiner Spalte“

Eine partielle Funktion μ ist eine **Lösung des Ausdrucks $BGP(T)$** (T : Liste von Tripeln), falls gilt:

- 1 Domäne von μ ist genau die Menge der Variablen in T
- 2 Durch Ersetzung von leeren Knoten durch URIs, leere Knoten oder RDF-Literale kann man T in eine Liste von Tripeln T' umwandeln, so dass gilt:

Alle Tripel in $\mu(T')$ kommen im angefragten Graph vor

Ergebnis von $BGP(T)$:

Liste aller solcher Lösungen μ (Reihenfolge undefiniert)

Vereinigung von Lösungen

Zwei Lösungen μ_1 und μ_2 sind **kompatibel** wenn gilt
 $\mu_1(x) = \mu_2(x)$ für alle x , für die μ_1 und μ_2 definiert sind

Vereinigung von zwei kompatiblen Lösungen μ_1 und μ_2 :

$$\mu_1 \cup \mu_2(x) = \begin{cases} \mu_1(x) & \text{falls } x \text{ in der Domäne von } \mu_1 \text{ vorkommt} \\ \mu_2(x) & \text{falls } x \text{ in der Domäne von } \mu_2 \text{ vorkommt} \\ \text{undefiniert} & \text{in allen anderen Fällen} \end{cases}$$

\rightsquigarrow einfache Intuition: Vereinigung von zusammenpassenden
Tabellenzeilen

Definition der SPARQL-Operationen

Jetzt können wir wesentliche Operationen definieren:

- $Filter(\Psi, F) = \{\mu \mid \mu \in \Psi \text{ und } \mu(F) \text{ ist ein Ausdruck mit Ergebnis } \text{true}\}$
- $Join(\Psi_1, \Psi_2) = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Psi_1, \mu_2 \in \Psi_2, \text{ und } \mu_1 \text{ kompatibel zu } \mu_2\}$
- $Union(\Psi_1, \Psi_2) = \{\mu \mid \mu \in \Psi_1 \text{ oder } \mu \in \Psi_2\}$
- $LeftJoin(\Psi_1, \Psi_2, F) = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Psi_1, \mu_2 \in \Psi_2, \text{ und } \mu_1 \text{ kompatibel zu } \mu_2 \text{ und } \mu_1 \cup \mu_2(F) \text{ ist ein Ausdruck mit Ergebnis } \text{true}\} \cup \{\mu_1 \mid \mu_1 \in \Psi_1 \text{ und für alle } \mu_2 \in \Psi_2 \text{ gilt: entweder ist } \mu_1 \text{ nicht kompatibel zu } \mu_2 \text{ oder } \mu_1 \cup \mu_2(F) \text{ ist nicht } \text{true}\}$

Legende:

Ψ, Ψ_1, Ψ_2 – Ergebnisse, μ, μ_1, μ_2 – Lösungen, F – Filterbedingung

Beispiel

```
@prefix ex:    <http://example.org/> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet      ex:Autor    ex:Shakespeare ;
               ex:Preis    "10.50"^^xsd:decimal .
ex:Macbeth     ex:Autor    ex:Shakespeare .
ex:Tamburlaine ex:Autor    ex:Marlowe ;
               ex:Preis    "17"^^xsd:integer .
ex:DoctorFaustus ex:Autor  ex:Marlowe ;
               ex:Preis    "12"^^xsd:integer ;
               ex:Titel    "The Tragical History of Doctor Faustus" .
ex:RomeoJulia  ex:Autor    ex:Brooke ;
               ex:Preis    "9"^^xsd:integer .
```

```
{ ?buch    ex:Preis    ?preis .  FILTER (?preis < 15)
  OPTIONAL { ?buch    ex:Titel    ?titel . }
  { ?buch    ex:Autor    ex:Shakespeare . } UNION
  { ?buch    ex:Autor    ex:Marlowe . }
}
```

Beispielrechnung (1)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch
ex:Tamburlaine
ex:DoctorFaustus

Beispielrechnung (2)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
             BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch
ex:Macbeth
ex:Hamlet

Beispielrechnung (3)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                    <http://eg.org/Marlowe>.) )  
  )  
)
```

buch
ex:Hamlet
ex:Macbeth
ex:Tamburlaine
ex:DoctorFaustus

Beispielrechnung (4)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	preis
ex:Hamlet	"10.50"^^xsd:decimal
ex:Tamburlaine	"17"^^xsd:integer
ex:DoctorFaustus	"12"^^xsd:integer
ex:RomeoJulia	"9"^^xsd:integer

Beispielrechnung (5)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
              BGP(?buch <http://eg.org/Autor>  
                  <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	titel
ex:DoctorFaustus	"The Tragical History of Doctor Faustus"

Beispielrechnung (6)

```
Filter((?preis < 15),
  Join(
    LeftJoin(
      BGP(?buch <http://eg.org/Preis> ?preis.),
      BGP(?buch <http://eg.org/Titel> ?titel.),
      true
    ), Union(BGP(?buch <http://eg.org/Autor>
      <http://eg.org/Shakespeare>.),
      BGP(?buch <http://eg.org/Autor>
      <http://eg.org/Marlowe>.)
    )
  )
)
```

buch	preis	titel
ex:Hamlet	"10.50"^^xsd:decimal	
ex:Tamburlaine	"17"^^xsd:integer	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical History ..."
ex:RomeoJulia	"9"^^xsd:integer	

Beispielrechnung (7)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
             BGP(?buch <http://eg.org/Autor>  
                 <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	preis	titel
ex:Hamlet	"10.50"^^xsd:decimal	
ex:Tamburlaine	"17"^^xsd:integer	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical History ..."

Beispielrechnung (8)

```
Filter((?preis < 15),  
  Join(  
    LeftJoin(  
      BGP(?buch <http://eg.org/Preis> ?preis.),  
      BGP(?buch <http://eg.org/Titel> ?titel.),  
      true  
    ), Union(BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Shakespeare>.),  
             BGP(?buch <http://eg.org/Autor>  
              <http://eg.org/Marlowe>.) )  
  )  
)
```

buch	preis	titel
ex:Hamlet	"10.50"^^xsd:decimal	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical History ..."

- 1 Einleitung und Motivation
- 2 Umwandlung von Anfragen in SPARQL-Algebra
- 3 Rechnen mit der SPARQL-Algebra
- 4 Zusammenfassung**

SPARQL als Anfragesprache für RDF

- W3C-Standard (beinahe), sehr große Verbreitung
- Anfrage basierend auf Graphmuster
- Diverse Erweiterungen (Filter, Modifikatoren, Ausgabeformate)
- Spezifikation von Anfragesyntax, Ergebnisformat, Anfrageprotokoll
- Semantik durch Übersetzung in SPARQL-Algebra